Software Engineering Stage 6 (Year 12)

Software Engineering Project

Project Work Requirement

Contents

[Unit overview 4](#_Toc183506413)

[Identifying and defining 4](#_Toc183506414)

[Research, planning and issues 4](#_Toc183506415)

[Producing and implementing 5](#_Toc183506416)

[Testing and evaluating 5](#_Toc183506417)

[Assessment task overview 6](#_Toc183506418)

[Project description 7](#_Toc183506419)

[Submission details 9](#_Toc183506420)

[Steps to success 10](#_Toc183506421)

[Glossary 13](#_Toc183506422)

[NESA glossary keywords 16](#_Toc183506423)

[The Software engineering project 17](#_Toc183506424)

[Personal interest 17](#_Toc183506425)

[Course review and preview 18](#_Toc183506426)

[Developing a project proposal 18](#_Toc183506427)

[The design and production process 20](#_Toc183506428)

[1. Identifying and defining 21](#_Toc183506429)

[Define and analyse problem requirements 21](#_Toc183506430)

[Assess the scheduling and financial feasibility 22](#_Toc183506431)

[Boundaries 23](#_Toc183506432)

[Explore tools to develop ideas and generate solutions 24](#_Toc183506433)

[Investigate types of software implementation methods 25](#_Toc183506434)

[2. Research and planning 27](#_Toc183506435)

[Project management 27](#_Toc183506436)

[Software development approaches 27](#_Toc183506437)

[Apply project management to plan and conduct the development and implementation of a project and software engineering solution 29](#_Toc183506438)

[Using collaboration tools 30](#_Toc183506439)

[Exploring the social and ethical aspects of software engineering projects 30](#_Toc183506440)

[Explore communication issues associated with project work 32](#_Toc183506441)

[Quality assurance 32](#_Toc183506442)

[Ensuring requirements are met using a continual checking process 32](#_Toc183506443)

[Compliance and legislative requirements 32](#_Toc183506444)

[Systems modelling 34](#_Toc183506445)

[Data flow diagrams 0](#_Toc183506446)

[Structure charts 2](#_Toc183506447)

[Class diagrams – OOP projects only (or part thereof) 4](#_Toc183506448)

[Storyboards 5](#_Toc183506449)

[Decision trees 6](#_Toc183506450)

[Algorithm design 7](#_Toc183506451)

[Explain the contribution of back-end engineering to the success and ease of software development 8](#_Toc183506452)

[3. Producing and implementing 10](#_Toc183506453)

[Design, construct and implement a solution to a software problem using appropriate development approach(es) 10](#_Toc183506454)

[Present a software engineering solution using presentation software 10](#_Toc183506455)

[Develop, construct and document algorithms 10](#_Toc183506456)

[Allocate resources to support the development of a software engineering solution 11](#_Toc183506457)

[Demonstrate the use of programmed data backup 11](#_Toc183506458)

[Implement version control when developing a software engineering solution 12](#_Toc183506459)

[Explore strategies to respond to difficulties when developing a software engineering solution 15](#_Toc183506460)

[Propose an ***additional*** innovative solution using a prototype and user interface (UI) design 16](#_Toc183506461)

[4. Testing and evaluating 17](#_Toc183506462)

[Apply methodologies to test and evaluate code 17](#_Toc183506463)

[Use a language-dependent code optimisation technique. 19](#_Toc183506464)

[Analyse and respond to feedback 21](#_Toc183506465)

[Evaluate the effectiveness of a software engineering solution 21](#_Toc183506466)

[Developing a report to synthesise feedback 22](#_Toc183506467)

[Developing a test plan 22](#_Toc183506468)

[Comparing actual output with expected output 23](#_Toc183506469)

[Testing data used/generated based on path and boundary testing 23](#_Toc183506470)

[Scenarios for students who do not have a client 25](#_Toc183506471)

[Scenario 1 – parking roster 25](#_Toc183506472)

[Scenario 2 – typing tutor game 27](#_Toc183506473)

# Unit overview

This unit of work guides students through the development of their Software engineering project. You will need to draw upon existing knowledge an skills from the year 11 course and new skills that are part of the Secure Software Architecture and Programming for the Web focus areas. Some skills will be covered by the teacher whilst you are completing you project. However, you can always ‘skip ahead’ and learn these skills yourself or rely on your previous expertise to complete that section of the project

## Identifying and defining

You will need to develop a project proposal. This should focus on the Programming for the We and Secure Software Architecture areas of development but may include OOP, different types of algorithms (Programming Fundamentals) or even Machine Learning. You may need to preview syllabus content for upcoming focus areas (Secure software architecture, Programming for the web and Software automation). You will need to:

* Identify, define and analyse the requirements of a problem (or opportunity).
* Explore tools used to develop ideas and generate solutions and investigate types of software implementation methods.
* Contact and interview a client as well as analyse software engineering case studies, scenarios and existing solutions.

## Research, planning and issues

* Research and use the Waterfall, Agile and WAgile software development approaches. They apply project management skills to plan and conduct the development and implementation of their software engineering solution.
* Explore social and ethical issues associated with project work, including working individually, collaboratively and responding to stakeholders.
* Explore communication issues associated with project work and investigate how software engineering solutions are quality assured.
* Demonstrate the use of modelling tools and explain the contribution of back-end engineering to the success and ease of software development.

## Producing and implementing

* Construct and implement a solution to a software problem or opportunity using (an) appropriate development approach(es).
* Construct and document algorithms, demonstrate the use of programmed data backup, implement version control and explore strategies to respond to difficulties when developing your solution.
* Propose an additional innovative solution using a prototype and user interface (UI) design.

## Testing and evaluating

* Apply methodologies to test and evaluate code.
* Use a language-dependent code optimisation technique, analyse and respond to feedback and evaluate the effectiveness of a software engineering solution.

The software engineering solution should be evaluated and tested to ensure its features meet the success criteria outlined in the problem definition and identified needs.

You will need to present your software engineering solution using presentation software to the class (and client) and submit the project documentation and code. This is part of Component C.

# Assessment task overview

**Type of task**: develop a software engineering project containing a solution, project documentation and a presentation.

**Outcomes being assessed**:

A student:

* justifies methods used to plan, develop and engineer software solutions **SE-12-01**
* applies structural elements to develop programming code **SE-12-02**
* evaluates practices to safely and securely collect, use and store data **SE-12-04**
* explains the social, ethical and legal implications of software engineering on the individual, society and the environment **SE-12-05**
* justifies the selection and use of tools and resources to design, develop, manage and evaluate software **SE-12-06**
* designs, develops and implements safe and secure programming solutions **SE-12-07**
* tests and evaluates language structures to refine code **SE-12-08**
* applies methods to manage and document the development of a software project  
   **SE-12-09**

[Software Engineering 11–12 Syllabus](https://curriculum.nsw.edu.au/syllabuses/enterprise-computing-11-12-2022) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2022.

**Weighting**: 30%

Students identify a real-world problem or opportunity that can be addressed through the development of a software engineering project. Students develop project documentation and engineer a software solution that addresses this real-world problem or opportunity. The software engineering project is presented to the class by simulating a client handover.

## Project description

****Students find a real or fictitious client for whom they can develop a software solution. Students unable to find a client may use Scenario 1 or Scenario 2 in the Student support material section of the assessment task.

**Possible clients include:**

* school-based positions: a teacher, librarian or school administrator
* a member of the wider community
* business owner
* a non-government organisation (NGO) that works on various humanitarian and social issues
* a sporting club, coach, manager or gym
* a PCYC, YMCA, Scouts, Guides or similar
* a community group, religious group or hobby group
* any other organisation or client approved by the teacher.

**Students meet and negotiate with their client to:**

* define and analyse requirements
* assess scheduling feasibility
* define boundaries
* continually check progress
* document all correspondence.

**Students should ensure:**

* their classroom teacher is involved in all contact and correspondence between themselves and their clients.
* that both course and client requirements are met.
* school-defined submission dates and client requirements will assist in determining the scale and scope of the final project.

**The software solution:**

* Be a web based solution (Programming for the Web) that:
  + Includes an interactive page with user inputs
  + Uses elements of front-end and back-end security features as detailed in the syllabus
  + Collects, stores and manipulates data in the back-end
  + Uses the SQL language for database manipulation
  + Uses scripting language (preferably JavaScript)
  + Uses core html and CSS description that are written by the student
  + Uses a recognised web development framework and server structure
* Be a solution that demonstrates the design, development and construction of algorithms
* Addresses all the ‘clients’ functional and performance requirements.

The solution should also:

* use a language-dependent code optimisation technique
* be configurable in terms of the interface layout, colour scheme, accessibility requirements, legislative and technical requirements as per client needs
* incorporate security elements such as usernames, passwords, basic encryption and decryption as per client requirements
* include a proposal for an additional innovative solution using a prototype and user interface (UI) design.

## Submission details

This is a single assessment task as detailed in the assessment booklet but students submit:

**Component A – project documentation**

Complete resource booklet including:

* process diary
* client correspondence and feedback
* Gantt chart
* use of modelling tools.

**Component B – software solution**

* Develop, code and upload a software solution for the problem as indicated in the project documentation.

**Component C – presentation**

* Present their software engineering project to peers in class
* Respond to Q&A.

## Steps to success

Table 1 – assessment preparation schedule

|  |  |
| --- | --- |
| Steps | What I need to do |
| Component A –  Project documentation | Ongoing completion of the Software Engineering project documentation booklet provided that involves 4 key stages.   1. Identifying and defining 2. Research and planning 3. Producing and implementing 4. Testing and evaluating |
| Identifying and defining | * Begin by defining and analysing the problem requirements including: * demonstrating need(s) or opportunities * assessing scheduling and financial feasibility * generating requirements including functionality (what it should do) and performance (speed) * defining data structures and data types * defining boundaries – in system and environment * Provide ongoing screen shots of tools used to develop ideas and generate solutions including: * brainstorming, mind mapping and storyboards * data dictionaries * algorithm design * code generation * testing and debugging * installation * maintenance. |
| Research and planning | * Apply ongoing project management to plan and conduct the development and implementation of the project including: * scheduling and tracking using a software tool, including Gantt charts * using collaboration tools * Explore communication issues associated with project work including: * involving and empowering the client * enabling feedback * negotiating * Investigate how software engineering solutions are quality assured including: * defining criteria on which quality will be judged * ensuring requirements are met using a continual checking process * addressing compliance and legislative requirements * Demonstrate the use of modelling tools including: * data flow diagrams * structure charts * class diagrams * decision trees. |
| Component B –complete software solution | * Develop and code a software solution for the problem as indicated in the project documentation * “Upload the solution” to Canvas |
| Producing and implementing | * Design, construct and implement a solution to a software problem using stated appropriate development approach(es) * Develop, construct and document algorithms * Implement version control when developing a software engineering solution * Propose an additional innovative solution using a prototype and user interface (UI) design. |
| Testing and evaluating | * Apply methodologies to test and evaluate code * Use a language-dependent code optimisation technique. |
| Component C – presentation | * Present their software engineering project to peers * Respond to questions and answers. |
| Testing and evaluating | * Analyse and respond to feedback * Evaluate the effectiveness of a software engineering solution. |

## Glossary

Many of the following words will gather more meaning to you as you work through this booklet. Each time you see a new word in bold throughout this workbook you can add its definition in the table below in case you need to refer back later.

Table 2 – glossary

|  |  |
| --- | --- |
| Word | Definition |
| agile | An iterative and flexible approach to software development, focusing on collaboration, customer feedback and small, rapid releases. |
| algorithm | A step-by-step procedure required to solve a problem. Algorithms may be presented in many ways, for example written instructions, flow charts or using a computer programming language. |
| app | A software program designed for a specific purpose to run on mobile devices or on a personal computer. An abbreviation of the word ‘application’. |
| Application Programming Interface (API) | An interface that allows an application or website to plug into another program or website. |
| boundary | An invisible line that separates the software system being built from everything else. It helps define the limits and expectations of a software system. |
| client | The person or organisation who requires the solution and is likely to directly use the software application. A person or organisation using the services of a software engineer. |
| code | The instructions that guide a program’s execution. |
| code optimisation | To modify existing code or design new algorithms to improve program execution speed, reduce memory usage and enhance overall system performance |
| computational thinking | A process in which a problem is analysed and solved so that a human, machine or computer can effectively implement the solution. It involves using strategies to organise data logically, break down problems into parts, interpret patterns and design and implement algorithms to solve problems. |
| data flow diagrams | Diagrams illustrating the flow of data within a system, showing how it is processed or transformed. |
| data structures | A data structure is a way of organising and storing data in a computer so that it can be accessed and used efficiently. |
| data types | Data types used in computing are expressed as either: string or text, character, integer, floating point or real, date and time and Boolean. |
| decision trees | Diagrams that represent decisions and their possible consequences, often used in decision-making processes. |
| end-user | The person or group who will ultimately use the system or product. |
| Gantt charts | Visual tools used for project management to represent the timing of tasks or activities. |
| iterative approach | A method of development where the project is divided into smaller parts and developed incrementally, with each iteration building upon the previous one. |
| Object-Oriented Programming (OOP) | A paradigm based on the concept of ‘objects’ that can contain data and code in the form of procedures. |
| Object-Relational Mapping (ORM) | A technique used in object-oriented programming to query and manipulate data from a database. |
| online collaboration | Working together on projects or tasks using internet-based tools and platforms. |
| outsourcing | Hiring external individuals or companies to perform tasks or services instead of internal employees. |
| procedural programming | A method of programming where the program is divided into functions. A program consists of data and procedures (modules) that operate on the data. Data and procedures are treated as separate entities. |
| prototyping | Building partial versions of a system to test ideas and gather feedback before full implementation. |
| pseudocode | A form of algorithm description that uses English-like statements with defined rules of structure and keywords. |
| system flowcharts | Visual representations of the flow of data or processes within a system. |
| verification and validation | Processes to ensure that the system meets specified requirements and functions correctly. |
| WAgile | WAgile is a hybrid of Waterfall and Agile Project Management |
| waterfall (structured) | A sequential development approach where progress flows steadily downward through predefined phases. |
| working remotely | Performing work from a location other than the traditional office setting, often enabled by technology and the internet. |

## NESA glossary keywords

NESA keywords can be used in the syllabus and in the Higher School Certificate examination. Familiarisation with these keywords can assist in understanding how to write and respond to questions.

|  |  |
| --- | --- |
| Key term | Definition |
| ****Apply**** | Use, utilise, employ in a particular situation. |
| ****Describe**** | Provide characteristics and features. |
| ****Explain**** | Relate cause and effect; make the relationships between things evident; provide why and/or how. |
| ****Investigate**** | Plan, inquire into and draw conclusions about. |

[Glossary of key words](https://www.nsw.gov.au/education-and-training/nesa/hsc/student-guide/glossary) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2024.

# The Software engineering project

## Course review and preview

Preview the [syllabus contjent](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview) for Programming for the Web for areas of interest and ensuring you cover the required areas listed in the Project Description.

Which of these topics integrate into the Software engineering project? Explain how in the space provided below.

|  |
| --- |
|  |

## Developing a project proposal

[Create a mind map](https://www.mindmaps.com/project-planning-with-mind-maps/) using a [Lucidchart](https://www.lucidchart.com/pages/?) (or similar tool) for your software engineering project proposal by linking the key concepts and findings from their:

* possible clients list
* personal interests
* Year 11 focus area projects review
* Year 12 focus areas preview.

The development of a mind map will assist you to identify and find connections between:

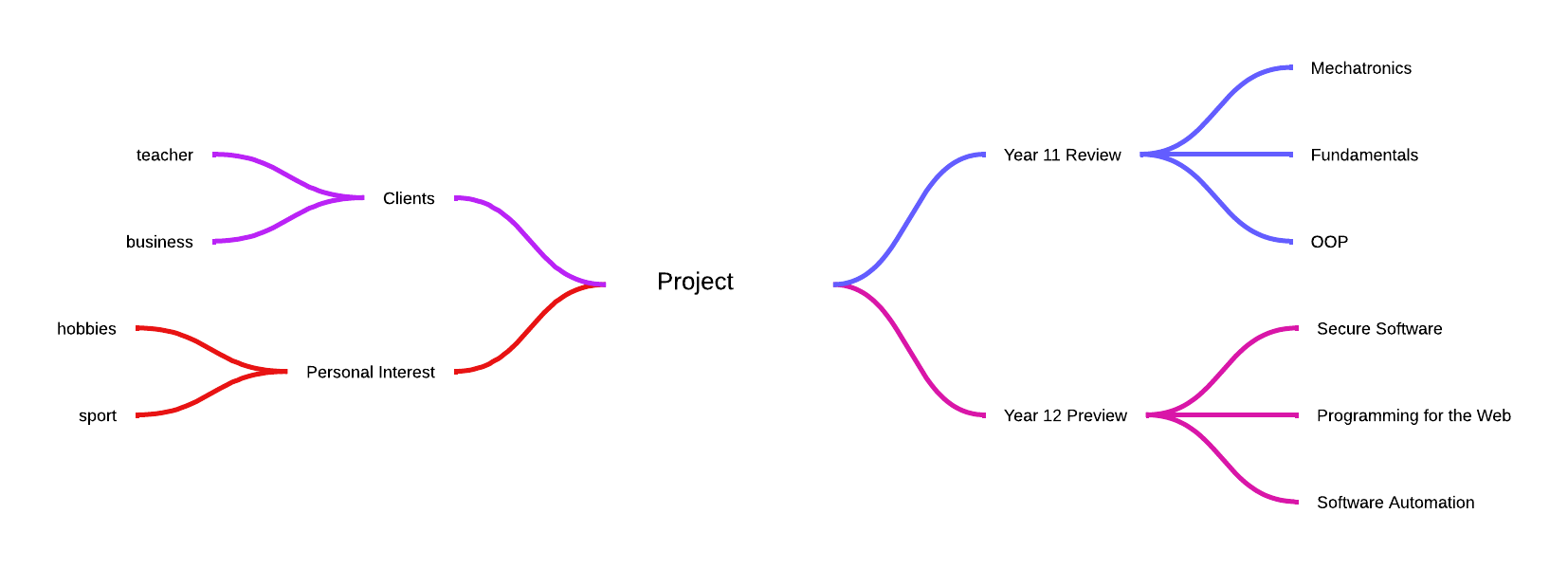
* what to plan to make
* who you will make it for
* how you will make it.

Figure 1 shows an incomplete mind map with 4 key aspects of the software engineering project: possible clients, personal interests, Year 11 focus areas and Year 12 focus areas.

* Extend each branch of the mind map as far as possible into individual ‘leaves’.
* Circle the ‘leaves’ that inspire or resonate most strongly.
* Find connections between ‘leaves’ from other branches.
* Annotate these connections into possible project ideas.
* Rank these ideas into a list of project preferences ensuring that a backup plan is identified.

Share your list with a critical friend, peer and, importantly, your teacher. Your proposal should be approved before continuing

Figure 1 – incomplete mind map using Lucidchart



## The design and production process

Throughout your study of Software engineering, you will learn about different types of design processes and how to apply them in your design project.

The design and production process:

* involves a sequence of organised steps which provide a solution to design needs and opportunities
* may take a few seconds or minutes, such as when you select what clothes to wear, or may take years as in the case with the design of a motor vehicle
* may involve one person or may involve many people
* may be simple or complex, depending on the task
* involves questioning (or evaluating) throughout the iterative process.

Figure 2 – flowchart of design and production process

Design and production process diagram
A flowchart labelled 'Ongoing evaluation' with a two-headed arrow indicating both directions. 
The first part of the flowchart is called '1. Identifying and defining'. It says 'identify and define the needs, opportunities and wants of a computing challenge, practise the technical skills, develop evaluation criteria.' There is an arrow pointing to the next section, which is labelled '2. researching and planning'. It says 'research, generate and practise ideas, be creative and propose new approaches to problems, explore new design opportunities.' An arrow points to the next section, labelled '3. producing and implementing', it says 'build and implement ideas, apply a variety of skills and techniques to create products that meet set criteria, modify and iterate solutions'. The arrow points to the next section, labelled '4. testing and evaluating'. It says 'test and evaluate solutions/products, evaluate quality and effectiveness against the criteria, make judgements throughout the solution and use these to refine the product.'
After testing and evaluating is a big arrow called 'Review if required to improve' and it goes all the way back up to the first part of the flowchart, indicating a cycle.

# 1. Identifying and defining

## Define and analyse problem requirements

“Discuss” with the client their needs and propose further opportunities. These will include the functional and performance requirements. In the table below describe the need(s) and opportunities that the software solution will provide for the client.

|  |  |
| --- | --- |
| Need | Opportunities |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Rank the needs and opportunities into priorities(‘must haves’) and opportunities (‘like to have’) by numbering the list.

### Assess the scheduling and financial feasibility

* Which of the needs and opportunities will be achieved first?
* Which are dependent upon the previous completion of another?
* Will there be any cost in this project?

In consultation with the client generate a ‘requirements of the problem’ statement that includes functionality (*Does it do what it is meant to do?*) and performance (*How well does it do it?*).

|  |
| --- |
|  |

Revise the use of data structures and datatypes from the Year 11 projects.

In consultation with the client, analyse the problem. The data structures and data types required will need to be defined. These are presented in data dictionaries when modelling the system.

List all of the entities (things) that will be represented and stored digitally in your software engineering solution and the data structures and data types that will be used to do so. For example:

* customers could be represented as an array of records or a database.
* high scores as integer variables.
* monster as an object from the character class.

### Boundaries

Boundaries define the limits of the problem or the system to be developed.

Defining the boundaries for the problem is essential so the client has realistic expectations of the limits of the new system.

Consider the software engineering solution and the aspects you can control.

Which parts of the problem remain outside the system (in the environment) that will interact with the system through an interface?

In the space below discuss any limitations or boundaries in which this new software will need to operate.

|  |
| --- |
|  |

## Explore tools to develop ideas and generate solutions

The following tools will gather meaning as you work through this booklet. Each time you use one of these tools take a screenshot and paste it into this workbook. Add a description of its use in the table below.

|  |  |  |
| --- | --- | --- |
| Tools | Screenshot | Description |
| Algorithm design |  |  |
| Brainstorming |  |  |
| Code generation |  |  |
| Data dictionaries |  |  |
| Debugging |  |  |
| Installation |  |  |
| Maintenance |  |  |
| Mind-mapping |  |  |
| Storyboards |  |  |
| Testing |  |  |

## Investigate types of software implementation methods

****Software implementation methods are:

* direct
* phased
* parallel
* pilot.

Each of these implementation methods has its advantages and disadvantages, and the choice depends on the specific context of the project, including its size, complexity and the organisation's risk tolerance.

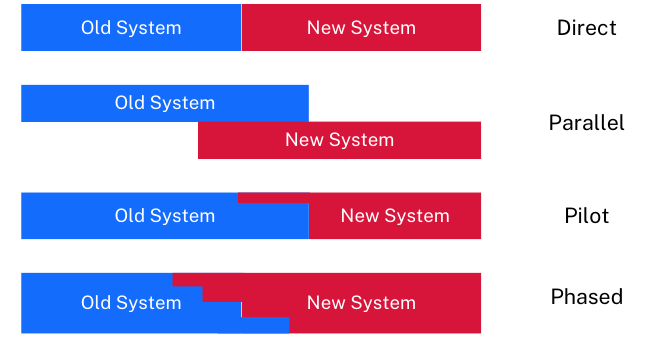
**Direct implementation** involves switching from the old system to the new system in one single action. All users move to the new system on a set date, and the old system is retired immediately. This method is straightforward and quick but can be risky if the new system has not been thoroughly tested.

**Parallel implementation** means both the old and new systems run simultaneously for a period. This allows for a comparison between the 2 systems to ensure that the new system operates correctly before the old system is decommissioned. While safer than direct implementation, running 2 systems in parallel can be resource intensive.

**Pilot implementation** involves rolling out the new system to a small, manageable group of users before a full-scale implementation. This method allows organisations to identify any issues or necessary adjustments in a controlled environment, reducing the risk of widespread problems.

**Phased implementation** involves gradually implementing the new system in phases or modules. Each phase is rolled out and stabilised before moving on to the next, allowing for incremental testing, training and adaptation. This method can reduce risk and disruption but may take longer to fully implement the new system.

Figure 3 – the 4 implementation methods.



In the space below **explain** the applicability of the implementation method for the current project.

|  |
| --- |
|  |

# 2. Research and planning

## Project management

### Software development approaches

**The Waterfall software development approach**

|  |  |
| --- | --- |
| Question | Sample explanation |
| 1.1 How are the logical progression of steps used throughout the life cycle? | [Insert sample explanation] |
| 1.2 What are the stages of ‘falling water’? |  |
| 1.3 What are the advantages and disadvantages of this approach. |  |
| 1.4 Give examples of the scale and types of developments that use this approach. |  |

**The WAgile software development approach**

|  |  |
| --- | --- |
| Question | Sample explanation |
| 2.1 Explain why it is a hybrid model | [Insert sample explanation] |
| 2.2 Analyse the ‘when’ intervention is applied during the development life cycle |  |
| 2.3 Analyse the ‘how’ intervention is applied during the development life cycle |  |
| 2.4 Give examples scale and types of developments that use this approach |  |

**The Agile software development approach**

|  |  |
| --- | --- |
| Question | Sample explanation |
| 3.1 What is the rate of developing a final solution? | [Insert sample explanation] |
| 3.2 Explain method tailoring |  |
| 3.3 Explain iteration workflow |  |
| 3.4 Give examples of the scale and types of developments that use this approach |  |

1. Include in your submission:

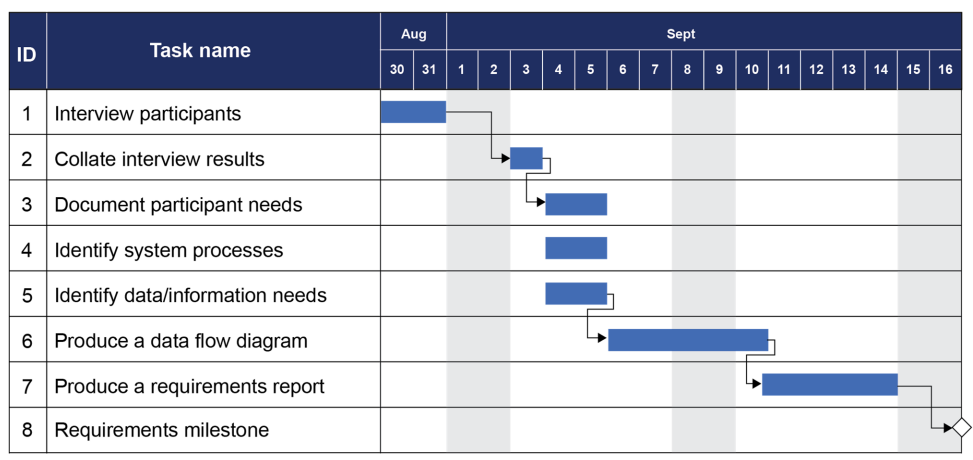
* a Venn diagram that shows the 3 Software development approaches similarities and differences.
* a justification of the use of one of these approaches for their current software engineering project.

### Apply project management to plan and conduct the development and implementation of a project and software engineering solution

#### Scheduling and task allocation

Students refer to the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://library.curriculum.nsw.edu.au/341419dc-8ec2-0289-7225-6db7f2d751ef/94e1eb0a-0df7-4dbe-9b72-5d5e0d17143a/software-engineering-11-12-higher-school-certificate-course-specifications.PDF) to develop a Gantt chart that details the tasks required to be completed, person or people assigned to each task, timeline that does not exceed the project due date and resources required. A sample Gantt chart is shown in the figure below.

Figure 4 – sample Gantt chart



Gantt chart from Higher School Certificate Course Specifications – Software Engineering (NESA 2023).

***\*\*\**** A Gantt chart is to be created and maintained by the student throughout the project\*\*\*

### Using collaboration tools

In the table below **identify** and describe collaborative tools used to manage this project and develop the solution, for example: Repl.it, GitHub, and so on.

|  |  |  |
| --- | --- | --- |
| Name | Description | Screenshot |
|  |  |  |
|  |  |  |
|  |  |  |

### Exploring the social and ethical aspects of software engineering projects

Consideration of social and ethical issues enables the development of software that not only meets technical requirements but also has a positive impact on society.

Complete a description for each key issue listed below. A ***sample*** answer has been provided in the table below.

|  |  |
| --- | --- |
| * Privacy | * Fairness |
| * Security | * Intellectual Property |
| * Accessibility and inclusivity | * Collaboration |
| * Transparency | * Feedback |

|  |  |
| --- | --- |
| Social and ethical aspects | Description |
| Privacy | Data collected and processed by software must be done so with the user's consent and comply with relevant regulations. |
| Security | Software must be developed with robust security measures to protect against cyber threats and data breaches. |
| Accessibility and inclusivity | Software must be accessible to all users, including being inclusive of those with disabilities, by following accessibility guidelines. |
| Transparency | Transparency around how software works, including any algorithms or AI systems used, builds trust with stakeholders. |
| Fairness | Software must not discriminate against any group or individual, and consideration needs to be given to the potential biases in algorithms. |
| Intellectual Property | Intellectual property rights must be respected including copyrights, patents and trademarks, when developing software. |
| Collaboration | Working collaboratively with team members and stakeholders ensure that the software meets their needs and addresses any concerns. |
| Feedback | Being open to feedback from stakeholders and being willing to make changes to the software based on their input ensures better results. |

### Explore communication issues associated with project work

You will need to be able to demonstrate both the correct and incorrect ways to do the following:

* involve and empower the client – treat the client with respect and dignity. They are not a developer so don’t talk to them in developer jargon. They don’t know what Python or SQL is. They just want a solution to they problem
* enable feedback – encourage regular feedback. This can be through messaging systems, email, regular meetings or phone calls but you will need to keep your teacher informed. We need to ensure your safety at all times. DO NOT meet with client on your own in person.
* Negotiate – your client isn’t aware of your student responsibilities. The scope of the project and boundaries need to be what is achievable in the timeframe and the limitations of the course content (your skills and knowledge). It may need to need to negotiate to do just part of what they need.

### Quality assurance

**Quality criteria**

Students **explain** quality criteria based upon the needs and functional requirements. These quality criteria should contain qualities, characteristics or components that need to be included or visible by the end of the current project.

Table 3 – criteria and explanation

|  |  |
| --- | --- |
| Quality criteria | Explanation |
|  |  |
|  |  |
|  |  |

### Ensuring requirements are met using a continual checking process

Students maintain Gantt or scheduling charts to indicate where continual checking process are ensuring requirements are met. Minutes of meetings, emails and diarised notes should be retained as evidence that the projects are on track. These can be kept in the log / diary.

Compliance and legislative requirements

Students **explain** compliance and legislative requirements their projects need to meet and how they plan to mitigate them where possible. For example, projects that deal with sensitive personal data being publicly available may fall under the [Privacy and Personal Information Protection Act 1998](https://legislation.nsw.gov.au/view/whole/html/inforce/current/act-1998-133" \l "statusinformation) (NSW) and/or [*Privacy Act 1988*](https://www.legislation.gov.au/Series/C2004A03712) (Cth). Alternatively, international standards on information security management such as [ISO/IEC 27001](https://www.iso.org/standard/27001) may also be applicable.

Table 4 – compliance and legislation

|  |  |
| --- | --- |
| Compliance or legislative issue | Methods for mitigation |
|  |  |
|  |  |
|  |  |

### 

## Systems modelling

Students are to **develop** the given tables and diagrams. Students should consult the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022) guide should they require further detail, exemplars or information. Each subsection below should be completed with reference to the functional requirements.

**Data dictionaries and data types**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Variable | Data type | Format for display | Size in bytes | Size for display | Description | Example | Validation |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Variable | Data type | Format for display | Size in bytes | Size for display | Description | Example | Validation |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

### Data flow diagrams

Data flow diagrams (DFDs) are used in software engineering to visualise the flow of data within a system. They are a graphical representation that shows how data is input into the system, processed and output. Complete the table below on the uses of Data flow diagrams.

|  |  |
| --- | --- |
| Uses of data flow diagrams | Description |
| Understand system requirements | DFDs help stakeholders, including developers, clients and users understand the flow of data within the system. They provide a clear and visual representation of how data moves through the system, which helps understanding system requirements. |
| Identify data sources and destinations |  |
| Define processes |  |
| Show data stores | DFDs show where data is stored within the system. This includes both temporary storage (for example, variables and buffers) and permanent storage (for example, databases and files). |
| Identify data flows |  |
| Analyse system behaviour |  |
| Document system design | DFDs serve as a documentation tool for system design. They provide a visual representation of the system's architecture, which can be used to communicate design decisions to stakeholders and developers. |

Read [What is a data flow diagram?](https://miro.com/diagramming/what-is-a-data-flow-diagram/)

|  |  |  |
| --- | --- | --- |
| Symbol name | Symbol | Description |
| Processes |  |  |
| Data stores |  |  |
| External entity |  |  |
| Data flow |  |  |

#### DFD levels

**Level 0 (Context diagram) –** this provides a high-level overview of the system, showing the system as a whole and its interactions with external entities.

**Level 1 DFD** **–** offers more detail than the context diagram by breaking down the system into major processes, showing how data flows between them and external entities. It provides a more granular look at the system's operation but keeps the focus on the overall system rather than minute details.

**Level 2 DFD and beyond –** these diagrams dive deeper into each process depicted in a Level 1 DFD, detailing the sub-processes and their data flows. The further you go beyond Level 2, the more detailed and focused the examination of processes and data flows becomes.

When creating a DFD, start with identifying major inputs and outputs, then build a context diagram (Level 0). Expand this into a Level 1 DFD by detailing major processes and how data flows between them. You can further detail these processes in a Level 2 DFD if necessary. Always ensure your diagram is accurate and easily understandable, checking with others for comprehension.

Examine examples of Level 0 and Level 1 DFDs in the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://library.curriculum.nsw.edu.au/341419dc-8ec2-0289-7225-6db7f2d751ef/94e1eb0a-0df7-4dbe-9b72-5d5e0d17143a/software-engineering-11-12-higher-school-certificate-course-specifications.PDF).

**Use specialised software (draw.io or similar) to create** [**Data Flow Diagrams**](https://miro.com/diagramming/data-flow-diagram/) **for the Software engineering project.**

### Structure charts

An example can be found in the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022).

****Constructing a structure chart to represent a library system involves identifying the main modules or components of the system and how they interact with each other.

#### Steps to create a structure chart

1. Brainstorm all the functions that need to be coded. This involves:
2. analysing the problem to determine what is needed
3. using a top-down approach to decompose (break) the problem into the sub functions
4. indenting can be used to show sub functions.
5. Construct the diagram. This involves:
6. drawing a box-rectangle for each function. Use verbs, for example, borrow, return, locate, check, update in each rectangle
7. drawing the interconnecting lines between the boxes, showing how the functions are linked to each other
8. adding symbols to the diagram
9. loops (for example for when a function is called multiple times)
10. case (diamond) – (choice of functions)
11. passing of parameters (open – variable, closed – control).
12. Provide explanations and further details to explain your diagram.

This could include providing a short description of each function and explanations of other aspects of your design that someone else reading your diagram would need to understand your design.

Use the following checklist when making a structure chart. Have you:

1. listed all the ‘sub functions’ for the system?
2. interconnected them correctly to relate the correct relationship between subs?
3. used the correct symbols to relate passing variables, loops and decisions?

***Students develop a structure chart demonstrating how the procedures, modules or components of their final software solution are interconnected.***

### Class diagrams – OOP projects only (or part thereof)

Class diagrams provide a visual representation of systems that are implemented using the object-oriented paradigm. They model classes, their attributes and methods and the relationships between classes.

Each diagram will include attributes and methods.

Class diagram(s) should be created for this project where an sections of the coded solution employ OOP. Class diagrams are not required if OOP is not used.

### Storyboards

A storyboard shows the various interfaces (screens) as well as the links between them. For example the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022) storyboard shows the relationship between 3 pages of information aimed at promoting a school canteen on a website.

**** A storyboard should be drawn (ideally on paper) to create a visual representation of the webpages and how they interact. It can be used in the initial discussions with the client to show ideas and possibilities.

**Develop a** storyboard in the space below, with at least 3 screens that visually represent the software solutions they will build.

### Decision trees

Examples can be found in the [Software Engineering HSC Course Specifications (PDF 2.9 MB).](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022)

D**evelop** decision trees to visually outline the logic flow and chain of decisions or selections the final solution will need. Students use the space below to develop a decision tree for the software engineering project.

|  |
| --- |
|  |

### Algorithm design

Software engineering students must be able to develop and interpret algorithms represented as pseudocode and flowcharts. Pseudocode is a text-based method of describing the logic in an algorithm. It makes use of capitalised keywords and indentation to show control structures used. Flowcharts are graphic based diagrams that represent algorithms and are read from top to bottom and left to right. Students should regularly practise making pseudocode from flowcharts and flowcharts from pseudocode.

****In designing complex algorithms, it is essential that you start with a clear, uncluttered mainline. The mainline should reference required subroutines, the details of which are shown in separate algorithms. Each subroutine should be concise and correctly make use of further subroutines for detailed logic.

All algorithms designed should adhere to the [Software Engineering HSC Course Specifications (PDF 2.9 MB).](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022)

Using the space below, students **develop** algorithms using methods such as pseudocode and flowcharts to solve the problem and meet the needs outlined in the Identifying and defining phase of the Software engineering project. These algorithms should explicitly include the variables from the data dictionaries created in the previous section.

### Explain the contribution of back-end engineering to the success and ease of software development

Backend engineering plays a crucial role in the success and ease of software development by providing the foundation and infrastructure that allows applications to function effectively and securely.

Complete the table below to explain how each aspect of backend engineering contributes to the success and ease of software development. Include in your table examples of how this applies to your solution.

[How the Backend works? Server, Application, Database, and API (2:40)](https://www.youtube.com/watch?v=gOghS3BmaxI)

|  |  |
| --- | --- |
| Terminology | Description |
| technology used | Backend engineering involves selecting and implementing the …… |
| error handling | Backend engineers implement robust error-handling mechanisms to ensure that the application can gracefully handle …… |
| interfacing with front end | Backend engineers design APIs (Application Programming Interfaces) to….. |
| security engineering | Security is a critical aspect of backend engineering. Backend engineers implement ….. |

# 3. Producing and implementing

## Design, construct and implement a solution to a software problem using appropriate development approach(es)

During this process students capture and include screenshots to document the development of their solution.

Each screenshot should include a caption that **explains** how it links to the:

* Component A of the assessment task
* Project Management development approaches

These screenshots are to be included in Component C of the assessment task – the class presentation.

## Present a software engineering solution using presentation software

Students choose presentation software for example, Microsoft PowerPoint, Google Slides, Sway, Prezi, Canva and so forth, to present the screenshots and captions. This is Component 3.

## Develop, construct and document algorithms

Students develop, construct and document algorithms used in the software engineering project. They refer explicitly to the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022) paper to do so. During the presentation of their software solution they refer to the algorithms developed to write their code.

## Allocate resources to support the development of a software engineering solution

During the development of a software engineering solution a careful allocation of resources increases the likelihood of a successful product. Complete the table below:

|  |  |
| --- | --- |
| Operation | Description |
| Define project scope |  |
| Identify required resources |  |
| Allocate personnel |  |
| Allocate technology |  |
| Allocate infrastructure |  |
| Allocate budget |  |
| Monitor and adjust |  |
| Manage risks |  |

Highlight the key steps that most relevant to your project.

## Demonstrate the use of programmed data backup

While developing solutions software engineers design and implement a backup strategy. Complete the following table to outline the steps in formulating a programmed data backup.

|  |  |
| --- | --- |
| Operation | Description |
| Identify backup requirements |  |
| Determine the backup frequency |  |
| Select backup tools |  |
| Implement backup strategy |  |
| Test backup process |  |
| Monitor backup performance |  |
| Demonstrate backup and restore |  |
| Document backup procedures |  |

Highlight the key steps of the programmed data backup that are most relevant to your project.

## Implement version control when developing a software engineering solution

Implementing version control is crucial when developing a software engineering solution to manage changes to the code, track progress and collaborate effectively.

Research three version control systems for software developers and complete the table below.

|  |  |
| --- | --- |
| Version control systems (VCS) for software developers | Description |
| ****Git****: |  |
|  |  |
|  |  |

In the space provided describe how you managed version control.

|  |
| --- |
|  |

## Explore strategies to respond to difficulties when developing a software engineering solution

In the space provided below **describe** strategies you explored to respond to difficulties when developing your solution including:

* looking for a solution online
* collaboration with peers
* outsourcing.

|  |
| --- |
|  |

## Propose an ***additional*** innovative solution using a prototype and user interface (UI) design

In the space below [sketch a wireframe](https://designshack.net/articles/graphics/what-is-a-wireframe/) of your additional innovative solution:

|  |
| --- |
|  |

# 4. Testing and evaluating

## Apply methodologies to test and evaluate code

Software engineers apply various methodologies and techniques to test and evaluate code to ensure its quality, reliability, and functionality.

Research each of these methodologies and report how they relate to the testing of yoru solution including how the client should be involved:

* functional testing
* acceptance testing
* live data
* simulated data
* beta testing
* volume testing.

Suggest an analogy to complete the table of common methodologies.

|  |  |
| --- | --- |
| Methodologies to test and evaluate code | Descriptive analogy |
| testing | [Insert descriptive analogy.] |
| functional testing |  |
| acceptance testing |  |
| live data testing |  |
| simulated data testing |  |
| beta testing |  |
| volume testing | . |

Highlight the methodologies to test and evaluate code that are most relevant to your project.

## Use a language-dependent code optimisation technique.

Software engineers use various language-dependent code optimisation techniques to improve the performance, efficiency, and maintainability of their code. These techniques are specific to the programming language being used and take advantage of language features and characteristics.

The table below provides some common language-dependent code optimisation techniques.

|  |  |
| --- | --- |
| Common language-dependent code optimisation techniques | Description |
| C/C++ optimisations | Use of inline functions to reduce function call overhead.  Compiler optimisations such as loop unrolling, constant folding, and dead code elimination.  Efficient memory management using techniques like stack allocation and memory pooling. |
| Java optimisations | Just-In-Time (JIT) compilation to convert Java bytecode into native machine code at runtime for better performance.  Use of the Java Virtual Machine (JVM) optimisations such as HotSpot to dynamically optimise code based on runtime profiling. |
| Python optimisations | Utilising built-in functions and data structures (for example, list comprehensions, dictionaries) for more efficient code.  Using libraries like NumPy for numerical computations, which are optimised for performance. |

****Code optimisation techniques. The code optimisations you use in your project will be specific to the languages you employ in your project. You should detail your use of optimisation techniques in a table or tables similar to those below. These example tables will give you an idea of the types of optimisation available in languages

|  |  |
| --- | --- |
| C/C++ optimisations | Explanation and sample code |
| Use of inline functions to reduce function call overhead. | [Insert explanation and sample code] |
| Compiler optimisations such as loop unrolling, constant folding, and dead code elimination. | [Insert explanation and sample code] |
| Efficient memory management using techniques like stack allocation and memory pooling. | [Insert explanation and sample code] |

|  |  |
| --- | --- |
| Java optimisations | Explanation and sample code |
| Just-In-Time (JIT) compilation to convert Java bytecode into native machine code at runtime for better performance. | **[Insert explanation and sample code]** |
| Use of the Java Virtual Machine (JVM) optimizations such as HotSpot to dynamically optimise code based on runtime profiling. | **[Insert explanation and sample code]** |

|  |  |
| --- | --- |
| Python optimisations | Explanation and sample code |
| Utilising built-in functions and data structures (for example, list comprehensions and dictionaries) for more efficient code. | **[Insert explanation and sample code]** |
| Using libraries like NumPy for numerical computations, which are optimised for performance. | **[Insert explanation and sample code]** |

## 

## Analyse and respond to feedback

Students analyse feedback given to them on the software solution they have just created. This feedback can be in the form of an interview, survey, focus group, observation of the Q&A forum and may be from the client, peer review, critical friend or the teacher.

## Evaluate the effectiveness of a software engineering solution

Analysis of a solution against quality success criteriaStudents are to take each quality success criteria from Section 2 and place it in the table below. For each quality criteria, **analyse** the components of the solution that met or did not meet each quality criteria. Give reasons why each success criteria were or were not met.

|  |  |  |
| --- | --- | --- |
| Quality criteria | Met? | Analysis |
|  |  |  |
|  |  |  |
|  |  |  |

### Developing a report to synthesise feedback

Students collate all feedback and synthesise this into a report that categorises common themes including:

* usability
* performance conclusions
* recommendations.

In the space provided below **provide a summary of the feedback** that includes overall positive, and negative sentiments towards their software solution in their response.

|  |
| --- |
|  |

### Developing a test plan

Developing a test plan is crucial for evaluating the effectiveness of a software engineering solution by ensuring that the software is tested thoroughly, consistently, and in accordance with its requirements.

The test plan is a living document that can be updated and refined throughout the software development lifecycle. Software engineers use feedback from testing to update the test plan and continuously improve the effectiveness of the software engineering solution.

A test plan should:

* identify the resources required for testing, such as testing tools and environments
* define the criteria for evaluating the results of the tests, such as pass/fail criteria, acceptance criteria, and performance. This provides a basis for determining whether the software meets its requirements.
* include detailed test cases that describe the steps to be taken, the expected results, and the criteria for determining success or failure ensuring that the tests are conducted consistently and accurately
* specify the schedule and sequence for executing the tests. By following the test plan, testers can ensure that all necessary tests are conducted and that the results are recorded and analysed.

### Comparing actual output with expected output

Once the tests are completed results are analysed. This involves comparing the actual results to the expected results, identifying any discrepancies, and determining the cause of any failures.

The test plan includes a reporting mechanism for documenting the results of the tests. This helps stakeholders understand the effectiveness of the software engineering solution and any areas that may require further improvement.

### Testing data used/generated based on path and boundary testing

Students **identify** variables and values which were used for either path and/or boundary testing.

Students **develop** these test data tables based on their algorithms versus their real code.

Students then **state** the reason for including these variables.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Variable | Maximum | Minimum | Default value | Expected output | Actual output | Reason for inclusion |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Boundary testing comparing actual output with expected output.

## Scenarios for students who do not have a client

Scenarios where clients could be simulated by the teacher or classmates could include:

* online prefect voting system
* canteen ordering app
* market day business studies website
* school musical booking system
* a biometric roll call attendance system (controversial)
* school jersey and name design app
* sports carnival administration system
* event management system
* an interactive school map and timetable.

These should be expanded into detailed scenarios to provide guidance for students to:

* extract functional requirements
* model with tools
* develop software solutions and innovative prototypes.

The following 2 scenarios demonstrate the depth of detail required to identify, define and analyse to produce and implement a software engineering solution.

**Be aware that these given scenarios will make it more difficult to meet the criteria for the project work part of the assessment.**

### Scenario 1 – parking roster

Warami High School has an exceedingly small staff car park. They have a total of 30 spaces allocated for 120 staff members (including Student Administration and Support), split across the 10 faculties of:

* English – 15 teachers
* Mathematics – 13 teachers
* Science – 10 teachers
* Technological and Applied Studies (TAS) – 12 teachers
* Creative and Performing Arts (CAPA) – 7 teachers
* Personal Development, Health and Physical Education (PD/H/PE) – 7 teachers
* History and Languages – 7 teachers
* Human Society and Its Environment (HSIE) – 9 teachers
* Learning Support – 6 teachers
* Student Administration and Support – 34 staff members

The number of teachers for each faculty is given above. There are a total of 10 files provided, one representing each faculty, that need to be read into the program. In each file, there are the names of each of the teachers.

A timetable of how many spaces each faculty gets per day as well as who gets what spot each day for a 2-week cycle needs to be generated.

The program **must**:

* present a form for the user to select the files containing the names for each faculty
* read the names of each of the teachers in each of the faculties from the files
* display which faculty has what spaces on particular days linked to the teachers’ names, using a graphical method such as a block of colour or a picture of a car in a car park.
* enable the user to click on ‘Monday’ and get the allocations for Monday, or ‘Tuesday’ for Tuesday’s allocations and so on.

Note: Teachers do not have to be allocated the same days in a row, nor do they have to have the same parking spots each time.

The program **should** also:

* ensure that 50% of each faculty is allocated a parking spot in the fortnight
* give the user the option to either read the names of each teacher from the files, or to enter each teacher’s name manually with the number of days each teacher is in per fortnight
* have a menu system that allows the user to switch between the default mode as specified in the ‘must’ section, and this optional mode.

The program **could** also:

* ensure that 70% of each faculty is allocated a parking spot in the fortnight
* allow the user to change the number of staff members per faculty so long as the total of 120 staff members is reached. This means that the user should be presented with a form with each faculty name and an associated input box with how many staff members are in each faculty
* increase or decrease the number of total staff members in the school car parking timetable. This should in turn check to see that the total entered does not exceed the number of total staff members.

### Scenario 2 – typing tutor game

There is a lot of vocabulary in the Software Engineering course. There are also a lot of large, complex assessment tasks like this one which require a lot of typing. A typing game which builds up the vocabulary of the students studying Software Engineering, as well as keyboard skills, is required to bridge this gap. Technical vocabulary for this typing program is to be derived from the topics of:

* Programming fundamentals – 10 pieces of technical vocabulary
* Object-oriented programming – 10 pieces of technical vocabulary
* Programming mechatronics – 10 pieces of technical vocabulary
* Secure software architecture – 10 pieces of technical vocabulary
* Programming for the web – 10 pieces of technical vocabulary
* Software automation – 10 pieces of technical vocabulary

In addition, definitions and exemplar paragraphs for each of the below NESA key terms need to be included in the typing tutor program.

* Identify
* Define
* Describe
* Explain
* Analyse
* Justify
* Evaluate

These exemplar paragraphs are to be read from data files provided.

Additional exercises that train students on how to place their hands on the home row, feel for the bumps on F and J, punctuation marks and so on must also be included. Some examples of typing programs are given here:

* Speed Typing Online [(https://www.speedtypingonline.com/typing-tutor)](https://www.speedtypingonline.com/typing-tutor)
* Typing Trainer [(https://www.typingtest.com/trainer/)](https://schoolsnsw-my.sharepoint.com/personal/renee_cobcroft2_det_nsw_edu_au/Documents/Desktop/987-988-989-tas-software-engineering/working%20files/(https:/www.typingtest.com/trainer/))
* Typing Academy [(https://www.typing.academy/)](https://schoolsnsw-my.sharepoint.com/personal/renee_cobcroft2_det_nsw_edu_au/Documents/Desktop/987-988-989-tas-software-engineering/working%20files/(https:/www.typing.academy/)).

The program **must**:

* read the provided data file for the NESA key term exemplar paragraphs
* progress the students, from training them how to place their hands on the home row to typing in key words, then whole sentences and paragraphs
* calculate their words per minute and percentage (%) of accuracy
* contain at least 5 levels, from absolute beginner at Level 1 to typing in whole sentences and paragraphs at Level 5.

The program **should** also:

* give the administrator of the typing tutor the option of entering the technical vocabulary for each of the given Software Engineering topics into a form, which then gets written into at least one file
* read the technical vocabulary for each topic from the data file generated by the program.

The program **could** also:

* contain up to 10 levels, slowly progressing from absolute beginner at Level 1 to expert at Level 10, where the students are typing in exemplars of code that perform various tasks from Object-oriented programming such as
* class definition and object instantiation
* polymorphism
* inheritance and/or Secure software architecture to create MD5 hashes such as those in [Python](https://www.geeksforgeeks.org/md5-hash-python/) or in [JavaScript](https://ourcodeworld.com/articles/read/1547/how-to-create-md5-hashes-in-javascript)
* AES encryption and decryption ciphers in [Python](https://medium.com/quick-code/aes-implementation-in-python-a82f582f51c2), or PGP encryption and decryption in [Python](https://medium.com/@almirx101/pgp-key-pair-generation-and-encryption-and-decryption-examples-in-python-3-a72f56477c22) and [OpenPGP in JavaScript](https://github.com/openpgpjs/openpgpjs) as typing practice
* read these code exemplars from data files made for the program.